

Courant Institute of
Mathematical Sciences

AEC Computing and Applied Mathematics Center

A Computer Program for Doing
Tedious Algebra (SYMB66)

Arnold Lapidus, Max Goldstein,
and Susan S. Hoffberg

AEC Research and Development Report

Mathematics and Computers

January 1968

New York University

C-1



UNCLASSIFIED

AEC Computing and Applied Mathematics Center
Courant Institute of Mathematical Sciences
New York University

Mathematics and Computers

NYO-1480-88

A COMPUTER PROGRAM FOR DOING TEDIOUS ALGEBRA (SYMB66)

Arnold Lapidus, Max Goldstein, and Susan S. Hoffberg

Contract No. AT(30-1)-1480

UNCLASSIFIED

NEW YORK UNIVERSITY
COURANT INSTITUTE LIBRARY

Table of Contents

	Page
I. Introduction.....	1
II. Instant SYMB66.....	3
A Simple Example.....	8
III. Notation.....	10
A. Concept of an Expression.....	10
B. Input.....	14
C. Output.....	16
IV. The Algebraic Computation Package of Routines...	17
A. Package of Routines.....	17
1. List Processing Routines.....	18
2. Expression Manipulation Routines.....	20
3. Term Manipulation Routines.....	24
4. Operational Routines.....	25
5. Subalgebraic Routines.....	28
B. An Illustration.....	30
V. Preprocessor (QUEST).....	31
A. Format of Statements.....	31
B. Error Messages.....	32
C. Instructions for QUEST Programming.....	33
D. QUEST Statements.....	34
E. QUEST Instructions.....	35
F. QUEST Executable Type Statements.....	36
G. Examples of QUEST Statements.....	41
References.....	45

I. Introduction.

In a previous paper [1], we reported on a simple experimental algebraic manipulator which consisted of a set of subroutines for doing elementary algebra with expressions which were essentially polynomials. The information gained from experience with that system, and from reports of other workers in the field [3,4], led us to design an enlarged system which would include arbitrary functions of one variable and differentiation capabilities as well as a more problem-oriented appearance.

The original program (Symbolang) was rewritten using a revised internal representation of expressions. The new package forms the basis for the current system, called SYMB66. These routines still use a subset of the list processor SLIP, operate primarily on Fortran-like expressions, and allow various forms of symbolic algebraic manipulation. Superimposed on these routines is a pre-processor, so that a "non-expert" problem oriented user might not have to be concerned with the internal machinations of the list processor. This allows the use of a more natural problem description.

Those interested in using this higher level problem oriented language will find a telegraphic description of it in Section II, and more complete details in Section V.

Sections III and IV deal with specific details of the system. Section III discusses machine level notation, input and output, and Section IV discusses the algebraic computation package of routines.

II. Instant SYMB66

A preprocessor is available for SYMB66 which relieves the user of the task of learning the internal structure and other minutiae of the system. A summary of simplified commands is given followed by a detailed explanation of the usage and function of the processes. Section V gives more comprehensive information about the preprocessor.

<u>COMMAND</u>	<u>Abbrev. (if any)</u>	<u>Comment</u>
RESERVE	---	Reserves space
EXPRESSION	EXP	Creates Expression
NAME	---	Creates null expressions
THE DERIVATIVE FØR	TDF	Defines derivatives
ADD	---	Adds expressions
SUBTRACT	SUB	Subtracts expressions
DIVIDE	DIV	Divides by a variable
MULTIPLY	MUL	Multiplies expressions
DIFFERENTIATE	DIF	Differentiates expressions
EVALUATE	EVA	Substitutes constants
GET CØEFFICIENT...	GCØ	of variables to powers
GET PØWER...	GPØ	of specified variable
TRUNCATE	TRU	Truncates series
SØLVE	---	Solves simple linear eq.
SUBSTITUTE	SUB	See detailed description.
REPLACE	REP	another type of substitution
ØUTPUT	ØUT	Prints expressions
EMPTY	---	Clears expression
DESTRØY	DES	Annihilates expressions
CØPY	---	Makes copies of expression

SIMPLIFY	SIM	Collects terms
IF	---	Used in tests
AND PLACE ØN	APØ	Used in other commands
AND TRUNCATE ØN	ATØ	Another substitution option
EXCLUDING FUNCTION		
ARGUMENTS	EFA	Used in substitution
WITH RESPECT TØ	WRT	Used in DIF command.

RESERVE n - Allocates n memory locations for symbolic manipulation. Must be used first.

EXPRESSION la = "Fortran type expression" (use of "/" prohibited) - Creates a symbolic expression la .

NAME la, lb, lc, ... (up to 5 var) - Designates names of expressions which have the value zero until other terms are added to them.

THE DERIVATIVE FOR "function name", IS la = "expression for derivative". - Defines the derivative of the function named to be the expression for derivative which is to be an expression with the convention that the argument of the function is replaced by zero in the expression.

ADD la, TO lb - Adds the expression la to lb and destroys la.

SUBTRACT la, FROM lb - Subtracts la from lb and destroys la.

DIVIDE la, BY char, AND PLACE ØN lb - Divides an expression by a variable.

MULTIPLY la , BY lb , APØ lc - Multiplies the expression la by the expression lb and adds the results to lc .

MULTIPLY la , BY con - Multiplies the expression la by the constant con and leaves the result on la . la is the original and resulting expression.

The last five instructions may be written in the form

$lb = lb \pm la$ (for ADD, SUB)

$lb = la / char$ (for DIV)

$lc = la * lb$ (for MUL)

$la = la * con$ (for Mult by const)

provided that an S appears in column 1.

DIFFERENTIATE la , WITH RESPECT TO $char$, APØ lb -

Differentiates an expression la with respect to the variable $char$ and places the result on the expression lb .

EVALUATE la , FØR $char = X$, $char = X$, ... (up to 5) -

Substitutes the indicated numbers X for the indicated variables $char$ on the expression la and leaves the result on la .

GET CØEFFICIENT ØF $char ** e$, IN la , APØ lb - This is used to get the coefficient of a variable ($char$) to a power. If $**e$ is omitted the exponent is assumed to be 1.

GET PØWER ØF $char$, IN la , APØ var - This is used to find the power of a variable $char$ in the expression la . The result is stored in " var ." Ordinarily $char$ would appear

in only one term of the expression la .

TRUNCATE la , FOR char ** e - Eliminates terms in an expression la that contain char to a power greater than e. la is the original and resulting expression.

SOLVE la , FOR char, APO lb - Solves the equation $la = 0$ assuming that char appears linearly in only one term of the expression.

The next six explanations pertain to substitution.

REPLACE is used to mean insert an expression into the original in place of a designated character. SUBSTITUTE is used to mean insert the expression and then carry out the necessary multiplications.

SUBSTITUTE la , FOR char, IN lb , APO lc , EXCLUDING FUNCTION ARGUMENTS - Substitutes the expression la for char where it appears in lb except where it appears in the argument of a function.

SUB la , FOR char, IN lb , APO lc - Substitutes the expression la for char wherever it appears in lb ; even in the argument of a function.

SUB la , FOR char, IN lb , APO lc , EFA AND TRUNCATE ON var, pwr, var, pwr, ... (up to 5 pairs) - Substitutes the expression la for char where it appears in lb , excluding appearances in function arguments, and then eliminates all terms containing the variable var to a power greater than that specified. The results are put on lc .

SUB la , FOR char, IN lb , APØ lc , ATØ var, power, ...

(up to 5 pairs) - Substitutes the expression la for the variable char wherever it appears in lb and puts the results on lc after truncating terms in which var appears to a power greater than that specified in the sequence var, power, var, power, ...

REPLACE char, IN la , WITH lb , APØ lc - Simply replaces char wherever it appears in la by the expression lb .

The results are put on lc .

REP char, IN la , WITH lb , APØ lc , EFA - Replaces cher with lb where it appears in la except where it appears in a function argument. The results are added to lc .

OUTPUT $la, lb, \dots$ - Prints one to five expressions $la, lb, \dots$ in Fortran-like format.

EMPTY $la, lb, \dots$ (up to 5) - Eliminates all terms of an expression making the expression identically 0.

DESTROY $la, lb, \dots$ (up to 5) - Erases the expressions from memory.

COPY la , TØ lb - Makes a copy lb of the expression la .

SIMPLIFY $la, lb, \dots$ (up to 5) - Collects terms in an expression including function arguments.

IF la , CØNSTANT var = CON "statement number" - If the expression la is a constant, the value of the constant is stored in var and control transferred to the indicated statement number. Otherwise control is transferred to

the next statement.

IF ℓa , ZERO statement number - If ℓa is equal to zero, control is transferred to the statement indicated.

Otherwise control is transferred to the next statement.

IF ℓa , EQUAL ℓb , statement number - If ℓa is an exact copy of ℓb , control goes to the statement number specified; otherwise control goes to the next statement.

A Simple Example:

```
RESERVE 3500
PRINT 10
10  FORMAT(* THIS IS THE BEGINNING OF THE RUN* )
    NAME SINY, LTØZ
    EXP SINY = Y -.166 * Y **3 + .0083333*Y **5
    EXPLTØZ = 1 + Z + .5 * Z **2 + .166 * Z ** 3
    SUBSTITUTE SINY, FØR Z, IN LTØZ, APØ RESULT
    ØUTPUT RESULT
    CALL EXIT
    END
```

The computer output is:

THIS IS THE BEGINNING ØF THE RUN

```
SINY = Y - .166 *Y **3 + .0083333 *Y **5
LTØZ = 1 + Z + .5 * Z **2 + .166 *Z **3
RESULT = 1. + Y + .500000000 *Y **2 - .166000000 *Y **4.
        -.074334700 *Y **5 + .022111300 *Y **6. + .017872871 *Y **7.
        -.001383328 *Y **8 - .002137128 *Y **9. + .000034722 *Y **10.
        +.000148940 *Y **11- .000005741 *Y **13.+ .000000096 *Y **15.
```

Notes on Example:

<u>Line</u>	<u>Comment</u>
1	reserves space and sets up COMMON
2,3	FORTTRAN
4	creates null expressions named SINY and LØTZ (not necessary)
5,6	creates expressions -- an alternative way would be to omit EXP and insert X in col. 1
7	APO means "and put on"; RESULT is created explicitly; SUBSTITUTE may be abbreviated as SUB
8	Prints expression RESULT
9,10	FORTTRAN

Since division is not allowed in input, a subexpression which occurs as a divisor should be written with an exponent of -1.

III. Notation.

A. Concept of an Expression.

From the user's standpoint, the basic unit of operation is the "expression" which is composed of one or more terms. While the collection of algebra routines is organized primarily to compute with these expressions, and the programmer generally need not concern himself with individual terms, the user may have interest in operating on specific terms and thus an explanation of the structure of expressions as they appear on SLIP lists [2] follows.

Representation of an Expression on Lists.

The algebraic manipulator is designed to deal with those expressions whose Fortran representation contains neither division sign nor variable exponentiation nor function with more than one argument. An example of such an expression is

$$(1) \quad 1.5 X^{3.1} Y \sin (X^2) \sin^{4.5}(Y^2) \cos (X+Y \cdot \sin(Z))+1.2 X^2 Z$$

These expressions are contained on lists of the SLIP list processor [2] in which data which are pointers to lists are immediately identifiable as such. The representation is as follows.

Expression - A list of pointers to sublists.

An expression is a sum of the terms represented on its sublists.

Term - A list containing

- (a) A constant
- (b) A sequence of pairs corresponding to variables raised to powers.
- (c) A sequence of triples - each corresponding to a function, its argument and a power;

where

- (a') The constant is a floating point number.
- (b') The first element of each of the pairs is a representation of a variable, e.g., a Hollerith character. The second element is a floating point number representing the exponent associated with the variable.
- (c') The first element of each of the triples is a list name. The sublist named contains the expression which is the argument of the function.
The second element is a representation of the function name.
The third element is a floating point number representing the exponent associated with the function.

In Figure 1, the list representation of the expression (1) is diagrammed.

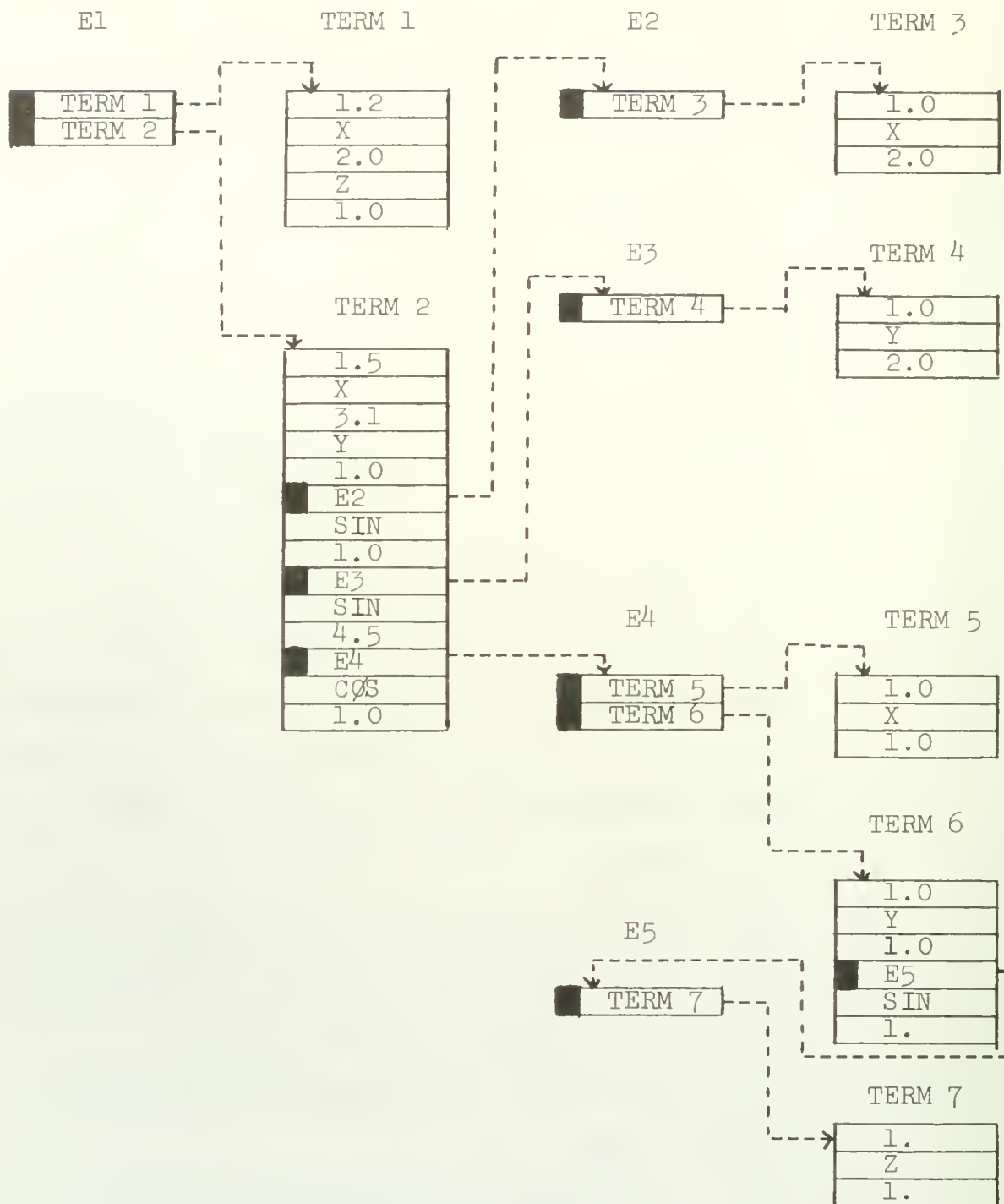


Figure 1. Representation of (1).

Heavy black mark indicates the element is a
sublist pointer.

Ordering within Terms.

Pairs within terms appear in the order of their associated variables, while the triples which appear within terms are ordered according to the function names. The ordering of variables is on the numerical value of the display code (on the 6600).

A Notational Trick.

Sometimes a user may want to work with an expression such as

$$x \cdot (x + y + z)^{200} .$$

This can be done by defining a function f with the property that $f(x) = x$. Then the above expression becomes

$$x \ f^{200}(x + y + z)$$

which is of the type easily represented by triples. In fact, a function f is already defined within SYMB66 and is called "l.*" instead of "f". The appearance of "l.*" in expressions is taken into account in the relevant parts of the code such as the output and derivative routines.

B. Input.

The most convenient method of introducing an algebraic expression into the program is by means of input data cards. This method is discussed here first. An alternative method is then discussed for creating expressions during the execution of the problem.

The expression to be inputted is punched onto cards just as if it were a Fortran expression (with continuation cards if necessary) and then followed by a card with an asterisk in column 1 (all other columns blank). Every expression must have a terminating * card. The cards are arranged as data in the order the programmer wishes the expressions to be introduced into his program. They may be interspersed with other data cards.

Then, whenever an expression is to be inputted into a program, the following statement appears in the coding

$$L = \text{LISTON}(SL)$$

where both L and SL refer to the same expression, as if SL appeared in an EQUIVALENCE statement with L.

The subroutine LISTON will read the Fortran-like expression on the next data cards and convert it into the SLIP list discussed in section A so that the expression will be available for symbolic computation. For

every expression that is inputted, a separate call to LISTON must be executed.

To create an expression consisting of a simple term which consists of a coefficient and a single variable to a power, i.e., of the form $C * Z^{**P}$ (where C, P are floating point numbers) and call the resulting expression L1; the sequence of statements is

```
L1 = LIST(9)
CALL PUTLST(L1,C,1HZ,P,L1,L1)
```

To create an expression $(CZ * FUNCT(L1))^{**Q}$ consisting of a simple term which consists of a coefficient C2 and a function to a power Q with an argument L1 (already our expression) and call it L2, the statements are

```
L2 = LIST(9)
CALL PUTLST(L2,C2,L1,5HFUNCT,Q,L2,L2)
```

From these simple expressions, arbitrary expressions usable by the system can be built up by performing multiplications, additions and substitutions.

Warning: An expression which is used as an argument of a function is then no longer available to the user.

If it is necessary to keep it, use

```
L8 = LSSCPY(L7)
```

L8 will then be an exact duplicate of L7.

C. Output.

The best, simplest, clearest and usual way to output an expression is to use the routine PRIPUT which prints expressions as card images of Fortran statements onto tape 6. A typical output statement is

```
CALL PRIPUT(6HRESULT,LA)
```

where the first argument is a display code name of the expression which appears to the left of the equal sign. The second argument is the name of the expression to be printed. Obviously, any BCD name can be assigned as the first argument.

There is another output routine which may be useful in unusual cases such as removing errors or displaying lists other than the usual algebraic expressions. The information on the list is printed in octal format in a vertical array. Such a format is hard to read and takes much space. For these reasons, it is not normally used. The calling sequence is

```
CALL PRLSTS(6HRESULT,LA)
```

exactly analogous to the use of the PRIPUT routine. It is a modified form of the SLIP routine of the same name.

Still a third way of obtaining output is to use the preprocessor statement OUTPUT described on page 7.

IV. The Algebraic Computation Package of Routines.

A. The package of routines is conveniently divided into five groups according to the types of operations they perform.

List processing routines: These are the SLIP routines which may be used by the programmer using the package.

Expression manipulation routines: Perform algebraic and arithmetic operations on expressions and return expressions as final results.

Term manipulation routines: Operate on terms of expressions. These are not generally needed by the programmer.

Operational routines: Perform non-arithmetic operations on expressions and terms.

All other routines in the package are called by the above routines. A list of these other routines along with a brief comment about each is given near the end of this section. The programmer need not concern himself with these subalgebraic routines.

1. List-processing routines (SLIP)

In the following, list operations are defined assuming that the lists are algebraic expressions. For a more general interpretation, see [2].

(i) CALL INITAS (array name, dimension of the array)

This call must appear in every program using this package of subroutines. The subroutine INITAS organizes the memory allocated by the user for storage of expressions. The memory requirement is determined by the user in a DIMENSION statement appearing before the CALL.

Example: DIMENSION ARR(1000)
 CALL INITAS(ARR,1000)

(ii) nom = LIST(9) is used to declare the name of an expression, where nom is the name. The expression is identically zero until terms are added to it.

(iii) CALL IRALST(nom)

When this statement is executed the expression designated by nom will be cleared from memory. All further references to the expression now will be meaningless and will thus cause the termination of the program.

(iv) CALL MTLIST(nom)

Makes the expression nom into one which is identically zero.

(v) new = LSSCPY(nom)

This statement, when executed, creates an element-by-element copy of nom which is then referred to by new. After execution, both new and nom are the same algebraic expressions although they occupy different locations in memory.

(vi) lval = LSTEQL(la,lb)

LSTEQL compares the expression la with the expression lb element-by-element. If the two expressions are identical, but perhaps in different parts of computer memory, then the value of LSTEQL is zero. Otherwise it is non-zero.

(vii) lval = LISTMT(nom)

LISTMT tests to see if the expression named nom is currently zero. If the expression is zero, lval = 0. Otherwise lval $\neq$ 0.

2. Expression Manipulation Routines.

The variables used in the given descriptions are designated by the following symbols to indicate the type of variable required or produced by the routine. The prefix "x" merely indicates the floating point named equivalent of the variable without the prefix (no conversion). All the routines may be used as functions or subroutines.

- (1) char Hollerith variable
- (2) fpo floating point number
- (3) lad the calling argument, which is the name of an expression to be operated upon, is not preserved, i.e. is destroyed.
- (4) lap the calling argument, which is the name of the expression to be operated upon, is preserved throughout operation.
- (5) lbp "
- (6) lco calling expression which is operated upon by the routine to become resulting expression
- (7) lcr resultant expression; created by routine if not already designated by programmer.

- (1) $\underline{x\ lco} = \text{ADD}(\underline{lad}, \underline{lco})$ adds $\underline{lad}$ to $\underline{lco}$
- (2) $\underline{x\ lcr} = \text{DVSUM}(\underline{lap}, \underline{char}, \underline{lcr})$ divides $\underline{lap}$ by $\underline{char}$ and adds result to $\underline{lcr}$
- (3) $\underline{x\ lco} = \text{EVALUE}(\underline{lco}, v_1, \text{con}_1, \dots, v_6, \text{con}_6)$ evaluates $\underline{lco}$ for the variable v_i equal to the constant con_i ($1 \leq i \leq 6$).

The following functions are automatically evaluated, when their arguments are constant; sin, cos, exp, log, tan, arctan and tanh.

- (4) $\underline{x\ lcr} = \text{GETCOE}(\underline{char}, \underline{fpo}, \underline{lap}, \underline{lcr})$ places the coefficient of $\underline{char} ** \underline{fpo}$ on $\underline{lcr}$.
- (5) $\underline{x\ lcr} = \text{HITENT}(\underline{lap}, \underline{char}, \underline{lbp}, \underline{lcr})$ replaces $\underline{lap}$ for $\underline{char}$ wherever it appears in $\underline{lbp}$ and adds results to $\underline{lcr}$. $\underline{lcr}$ results in factored form and no replacement is done in function arguments.
- (6) $\underline{lcr} = \text{INSBST}(\underline{lap}, \underline{char}, \underline{lbp}, \underline{lcr})$ substitutes $\underline{lap}$ for $\underline{char}$ wherever it appears in $\underline{lbp}$ and adds result to $\underline{lcr}$. $\underline{lcr}$ is not in factored form.
- (7) $\underline{lcr} = \text{INSUBT}(\underline{lap}, \underline{char}, \underline{lbp}, \underline{lcr}, v_1, p_1, v_2, p_2, \dots$
 (up to v_5, p_5)) substitutes as in (6) but eliminates all terms containing the variable $v_i ** d$, where v_i is a Hollerith variable and d is a floating point exponent $\geq p_i$.

- (8) lcr = INTENT(lap,char,lbp,lcr) replaces as in (5) but includes function arguments.
- (9) mc = LDIF(lap,char) differentiates lap with respect to char and creates the resulting expression mc. The derivatives for the following functions have been built into the package. The user may add other functions and their derivatives to the system for his own purposes by using the routine LOOKUP (see page 25).
- | | | | | | |
|--------|--------|--------|--------|--------|--------|
| sin | cos | exp | log | tan | |
| cot | sec | csc | arcsin | arccos | |
| arctan | artanh | arsech | arccot | arcsec | |
| arccsc | sinh | cosh | tanh | coth | |
| sech | csch | arsinh | arcosh | arcoth | arcsch |
- (10) lco = NUMPY(lco,fpo) multiplies lco by fpo.
- (11) xlcr = SBST(lap,char,lbp,lcr) substitutes as in (6) but does not consider function arguments.
- (12) xlcr = SOLVE(lap,char,lcr) assumes lap to equal zero (not identically), and that char appears linearly in exactly one term of lap. SOLVE solves for char and places result on lcr.
- (13) xlco = SUB(lad,lco) subtracts lad from lco.
- (14) xlcr = SUBT(lap,char,lbp,lcr,v₁,p₁,... (up to v₅,p₅)) substitutes as in (7) but does not consider function arguments.

- (15) x_{lcr} = SUMPY(lap,lbp,lcr) multiplies lap by lbp and adds results to lcr.
- (16) x_{lco} = TRCAL(lco,tchar,char) finds smallest power pwr of char in the terms in which the variable tchar is contained, and then truncates all terms with tchar to powers greater than pwr in lco.
- (17) x_{lco} = TRUNC(lco,char,fpo) eliminates all terms in lco whose power of char is greater than fpo.

3. Term Manipulation Routines.

- (1) $val = CPYTRM(\underline{la}, \underline{lb})$ If $\underline{la}$ is not empty, one term is removed from expression $\underline{la}$ to expression $\underline{lb}$ and $val = 1$. If $\underline{la}$ is empty, $val \neq 1$.
- (2) $\underline{xl} = DVTRM(\underline{l}, \underline{char})$ divides term $\underline{l}$ by $\underline{char}$ where $\underline{l}$ is the original as well as the resulting term $\underline{xl}$.
- (3) $\underline{xl} = SRTRM(\underline{l})$ arranges the variables of the term $\underline{l}$ (equiv. $\underline{xl}$) in lexicographical order. Then follows the variables with the functions which are also ordered.
- (4) $\underline{xl} = TRMMPY(\underline{m}, \underline{n}, \underline{l})$ multiplies term $\underline{m}$ by term $\underline{n}$ to make term $\underline{l}$, which is equivalent to $\underline{xl}$. If $\underline{l}$ has not been made a list name, TRMMPY also makes $\underline{l}$ a list name.

4. Operational Routines.

- (1) xla = BREAK(l,char,lb,la) separates the given expression l into expressions la and lb. Those terms which contain char are placed on lb, and those terms which do not, are placed on la (equivalent to xla). If la and lb have not previously been designated as expressions, BREAK declares them.
- (2) la = LEXICO(xla) collects the terms of the expression la (equiv. to xla) and also places them in a conventional lexicographical order.
- (3) la = LISTON(xla) creates a SYMB66 expression from a data card (see page 14).
- (4) CALL LOOKUP(-1.,fname,lder) allows the user to make available to LDIF, the differentiation routine, derivatives of functions used in his program, which are not already built into the package. (For a list of functions whose derivatives have been built in, see page 22). fname is the Hollerith name of the function for which the derivative is being specified. lder is the name of the "expression" created for the derivative of fname. Zero is always given in place of the argument of fname in

"expression" lder. For example to specify the derivative of tan (arg):

```
1  LL = LIST(9)
2  CALL PUTLST(0,LL,1.,0,3HSEC,2.,LL,LL)
3  CALL LOOKUP(-1.,3HTAN,LL)
```

Note in statement 2, zero is the first argument to PUTLST. This is required whenever PUTLST is used to create derivative expressions.

- (5) fpo = POWER(la,char) obtains the power fpo of the variable or function char where it first appears in the expression la.
- (6) CALL PRLSTS(Hollerith name, la) prints out any list designated la in octal format (see page 16).
- (7) xla = PRIPUT (Hollerith name, la) prints out any valid expression in Fortran notation (see page 16).
- (8) xla = PUTLST(la,a,b,c,...,la,la) adds terms to the expression la. The last term given in the calling sequence is followed by a double la, and each term is followed by la. See Section III on Notation.

- (9) xla = SMPL(la) orders the terms of the expression la within themselves and then collects the terms and places them in lexicographical order.
- (10) val = TSTC~~ON~~(la,con) tests if expression la is equal to a constant. If la is not equal to a constant val = 0. If la is equal to a constant, including the zero case, val = 1., and con is the constant.

5. Subalgebraic Routines.

(a) These routines are SLIP routines and descriptions for them can be found in [2].

LNKR	BØT	NAMTST	STRIND	SEQRDR
PØPTØP	AND	LNKL	PØPBØT	DELETE
ID	SEQLR	SEQLL	LCNTR	MAPØV
SETDIR	NUCELL	CØNT	RCELL	NEWTØP
STRDIR	INHALT	NEWBØT	LØCT	SETIND
TØP	EQUAL			

Some of these routines were carefully rewritten in ASCENT (CDC 6600 assembly language) which saved a little machine time. A more powerful technique has proven to be the overwriting of transfers to short closed routines with an equivalent open code. Doing this overwriting at execution time has proven to be both simple and efficient, yielding significant savings in central processor time.

(b) The first three below are list processing type routines:

- (1) LSCMPR compares the elements of two modified expressions for equality.
- (2) PØPMID deletes the list element which is currently being pointed to by the sequence reader and the value assigned to the function is the value of the element deleted.

- (3) MANY(nom,a,b,c,...) up to 59 variables are appended to the bottom of the list nom in the order the variables are given. nom must already be the name of a list.

The routines below are used by other routines in the package.

- (4) HREVAL is used by EVALUE
- (5) RAISE is also used by EVALUE; RAISE performs exponentiation
- (6) FØFX evaluates a function with a numerical argument.
- (7) EXTRAC determines if a given variable is present in a specified term, and if so, to what power.
- (8) KEBLA, FILL, NIPF are used by PRIPUT
- (9) KØKE is used by PUTLST
- (10) INLIST is an input routine used by the preprocessor QUEST
- (11) LCPYDR is used by LDIF to form the deriative of a function
- (12) SAME, MØNØ are used to make Fortran fixed-floating point conversions
- (13) ADDIN is used in lexicographical ordering
- (14) ERRPUT is used to print out error messages and abort execution.

B. An Illustration.

Suppose $\sin(x)$ is to be evaluated in its power series expansion for ten terms and $x = 2$.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + \frac{x^k}{k!} .$$

The coding for a complete program would be:

```
PROGRAM SINCH(INPUT,OUTPUT)
COMMON AVSL
DIMENSION ARR(1000)
COMMON /SYMSYS/SPQR(1000)
CALL INITAS(ARR,1000)
NSIG = -1.
NFAC = 1.
K = 19
LSIN = LIST(9)
DO 1 J=1,K,2
  NSIG = NSIG * (-1)
  IF(J.NE.1)NFAC = (NFAC+2)*(NFAC+1) * NFAC
  FJ = J
  FF = NSIG * (1./FLOAT(NFAC))
  LSN = LIST(9)
  CALL PUTLST(LSN,FF,1HX,FJ,LSN,LSN)
1 CALL ADD(LSN,LSIN)
  CALL PRIPUT(6HSIN(X),LSIN)
  CALL EVALUE(LSIN,1HX,2.)
  CALL PRIPUT(6HSIN(2.),LSIN)
  CALL EXIT
END
```

V. Preprocessor (QUEST)

A simple problem oriented language and preprocessor called QUEST (QUick and Easy Symb66 Translator) has been designed to facilitate the use of Symb66. This preprocessor translates into Fortran. Since the target language is Fortran, QUEST statements can be freely intermingled with Fortran statements. By coding in QUEST the programmer is relieved of the task of learning the internal structure of Symb66 and the various Fortran subroutine calls. For symbolic differentiation or substitution, the use of QUEST may be especially convenient.

The rules for the form of QUEST statements are given below followed by information on error messages and instructions for programming in QUEST. (A list of QUEST statements with their corresponding SYMB66 subroutine names and a sample QUEST program is given in Section II of this report (page 3).)

A. Format of statements.

- (1) Statement numbers and continuation cards are written in the usual Fortran manner, with up to 19 continuation cards allowed. Statements begin in column 7.
- (2) Blanks are ignored.
- (3) No variable identifier longer than ten characters is

accepted.

(4) All fixed point numbers are automatically converted to floating point numbers in QUEST statements.

(5) QUEST statements may be interspersed with Fortran statements, including Symb66 subroutine calls.

(6) Every variable identifier (char, con, name, pur, var, term, etc.) must be followed by a comma (,) or an equal sign (=), except where the variable identifier terminates an executable statement. (There is no terminal punctuation.)

(7) An X in column 1 indicates the Fortran-like arithmetic expression is a symbolic expression.

(8) An S in column 1 indicates that the desired arithmetic operations are symbolic.

B. Error messages.

Unacceptable QUEST statements are flagged with an error message. The errors recognized are:

comma	comma missing after an identifier
final punctuation	statement not correctly terminated
constant	constant incorrect or not found when expected
instruction	use of QUEST vocabulary incorrect
equal sign	equal sign missing when expected.

A listing of the QUEST program is always given to the programmer when errors are found, and the number of errors appears at the end of the listing. Also the message "QUEST DRØPS JØB" appears in the user's dayfile, and the user's entire job is aborted, so that no further processing takes place.

C. Instructions and Information for Programming in QUEST

(1) TAPE 2 must not be used by the programmer, and the variable names ABYSS, AVSL, SPQR, and SYMSYS.

(QUEST uses the file name TAPE 2 and the variable names.)

(2) A QUEST program begins and ends like a Fortran program. However the user must include the "RESERVE" statement followed by the total memory locations he has estimated for manipulating the algebraic expressions in his program. The RESERVE statement follows the header card.

(3) When the QUEST translator has processed a program successfully, the message "QUEST CØMPLETES JØB" appears in the user's dayfile. Subsequent control cards are then executed.

(4) All expressions appearing in the executable group of statements must have previously been declared (using EXP, X notation) or have been created in execution of a previous instruction. This latter case includes the

expression identifier following AND PLACE ON (APØ).

If these identifiers have not been previously declared to be expressions, the identifiers will be made into the desired expressions during execution of the statement, and will then be available for use.

(5) All QUEST expressions are preserved throughout execution of program except when specifically changed or destroyed. Exceptions to this statement are noted in Section II.

(6) QUEST abbreviations may be freely substituted for the longer forms of the statement. See Section II.

D. QUEST statements.

Preliminary comments on notation.

(1) All abbreviations are either underlined or appear in parentheses following the appropriate phrase. As a rule the longer commands are abbreviated to the first three letters of the word, and phrases of three words are abbreviated to a mnemonic consisting of the first letter of each of the three words.

(2) Several statements in both the executable group and the declarative group may be written as a standard type Fortran statement using the customary arithmetic operators. QUEST recognizes that these statements are

symbolic (rather than the usual Fortran) by an S (indicating operation) or X (indicating declaration) placed in column 1 by the programmer.

(3) In the examples below '*la*', '*lb*', and '*lc*' are all expression names and are variable. The SYMB66 or SLIP equivalent for each QUEST statement follows the semicolon.*

(4) The boldface characters are the elements of the QUEST language.

E. QUEST instructions.

Declarative-type statements

(1) To create a symbolic expression:

EXPRESSION *la* = Fortran type expression

(use of "/" operator prohibited)

or X in column 1 and merely the expression *la*
beginning in column 7;

LISTØN

(2) To designate the name of an expression, whose value is understood to be zero until other terms are added:

NAME *la, lb, lc, ...* (up to 5 var);

LIST

* The semi-colon is not part of QUEST.

(3) To add a derivative for a function not built into the SYMB66 system

THE DERIVATIVE FOR (TDF) function name, is
la = expression written with zero arguments
(no "/"'s);
LØØKUP.

(4) To allocate memory locations for symbolic manipulation:

RESERVE integer; CØMMØN AVSL and
CØMMØN/SYMSYS/SPQR(1000) and
DIMENSION ABYSS(n) and CALL INITAS (ABYSS,n)

F. Executable type statements.

a. Arithmetic operations

(1) To add:

ADD la, TØ lb; ADD - expression la is destroyed.

(2) To subtract:

SUBTRACT la, FRØM lb; SUB - la is destroyed.

(3) To divide an expression by a character:

DIVIDE la, BY char, AND PLACE ØN (APØ) lb; DVSUM.

(4) To multiply an expression by another expression:

MULTIPLY la, BY lb, APØ lc; SUMPY.

(5) To multiply an expression by a constant

MUL la, BY con; NUMPY - la is the original and
resulting expression.

The above instructions (1-5) may all be written in the following manner with an S in column 1:

$lb = lb \pm la$	ADD, SUB
$lb = la / \text{char}$	DIV
$lc = la * lb$	MUL
$la = la * \text{con}$	MUL

(6) To differentiate an expression with respect to a char; check that derivative is in system (see page 22) or use the TDF statement:

DIFFERENTIATE la , WITH RESPECT TO (WRT) char,
APØ lb ; LDIF.

(7) To evaluate a symbolic expression by giving a value to one or more variables:

EVALUATE la , FOR char = no, char = no,...
(up to 5 char = no); EVALUE - la is the original expression as well as the evaluated expression (for functions automatically evaluated see page 21).

(8) To obtain the coefficient of a character:

GET COEFFICIENT OF (GCØ) char ** no, IN la , APØ lb ;
GETCOE (if ** no is omitted the exponent is assumed to be 1).

(9) To obtain the exponent of a character:

GET POWER OF (GPØ) char, IN la , APØ var;
POWER (var is a variable).

(10) To eliminate terms in an expression that contain char to a power greater than no.

TRUNCATE la, FØR char **no; TRUNC la is the original and resulting expression; see (8) for explanation of char **no.

(11) To solve for char assuming the given expression is equal to zero, and that char appears linearly in only one term of the given expression:

SØLVE la, FØR char, APØ lb; SØLVE .

The next six statements (12-17) all pertain to substitution. To clarify the differences among the different statements, bear in mind that the word REPLACE, as it is used here, means to merely insert an expression into the original in place of a designated character, but not to carry out any multiplication. While on the other hand, SUBSTITUTE means to insert an expression for a specified character and to carry out all necessary multiplications in order to produce a simplified version of the original expression.

(12) To substitute an expression la for char appearing in lb, where appearances of char in function arguments is ignored:

SUBSTITUTE la, FØR char, IN lb, APØ lc, EXCLUDING FUNCTION ARGUMENTS (EFA); SBST .

(13) To substitute, as in (12), but also make substitutions for char appearing in function arguments:

SUB *la*, FØR char, IN *lb*, APØ *lc*; INSBST.

- (14) To substitute as in (12), but to eliminate all terms containing var to a power greater than the specified pwr:

SUB *la*, FØR char, IN *lb*, APØ *lc*, EFA AND TRUNCATE
ØN (ATØ) var, pwr, var, pwr, ... (up to 5 pairs);
SUBT .

- (15) To substitute as in (14) but to include function arguments as in (13):

SUB *la*, FØR char, IN *lb*, APØ *lc*- ATØ var,pwr,...
(up to 5 pairs); INSUBT .

- (16) To replace char whenever it appears in la:

REPLACE char, IN *la*, WITH *lb*, APØ *lc*; INTENT .

- (17) To replace char in la, but excluding appearances of char in function arguments:

REP char, IN *la*, WITH *lb*, APØ *lc*, EFA; HITENT .

b. Non-arithmetic expressions.

- (1) To print out one to five expressions:

ØUTPUT *la*, *lb*, ... ; PRIPUT .

- (2) To eliminate all terms of an expression, so that the expression is now equal to zero:

EMPTY *la*, *lb*, ... (up to 5); MTLIST .

- (3) To erase from memory an entire expression, so that even the name can no longer be referenced:

DESTROY *la*, *lb*, ... (up to 5); IRALST .

(4) To make a copy *lb* of the expression *la*, where *la* = *lb*
COPY *la*, TØ *lb*; LSSCPY .

(5) To simplify an expression, including its function arguments, by collecting terms:

SIMPLIFY *la*, *lb*, ... (up to 5); SMPL .

(6) To order the terms of an expression:

SØRT *la*, *lb*, ... (up to 5); LEXICØ .

This instruction is rarely used since lexicographical order is preserved by the SYMB66 system, and the user of QUEST ordinarily need not concern himself with this aspect.

Tests.

(1) Constant test - if the expression *la* is equal to a constant, the value of the constant is stored in the Fortran location *var*; in either case control may be transferred to the next statement; or if a statement no. appears, control is transferred to that statement if *la* is equal to a constant; if ELSE follows the statement no. control is transferred to the number following ELSE if *la* is not equal to a constant:

IF *la*, CØNSTANT *var* = CØN

(optional) statement no.

(opt.), ELSE statement no.; TSTCØN .

(2) Zero test - if *la* is equal to zero, control is transferred to statement no. following ZERØ; if *la* is

not equal to zero, control is either transferred to next statement or statement following ELSE if ELSE appears:

IF la , ZERO statement number (optional), ELSE
statement no.; LISTMT .

(3) Equality between two expressions - if la is an exact copy of lb control is transferred to given statement number; if la is unequal to lb , control goes to next statement, or statement following optional ELSE;

IF la , EQUAL lb , statement no. (optional),
ELSE statement no.; LSTEQL .

G. Examples of QUEST Statements .

The following merely demonstrates valid QUEST statements. Although the program was compiled, it is not intended to be mathematically meaningful.

```

PROGRAM MAIN(INPUT,OUTPUT)
RESERVE 3500
EXPLA=2+7.5*X +A*Y**2*Z
EXPRESSION LB = 15*X**2
COPYLA, TO LC
ADD LA, TO LB
CALL PRIPUT(2HLB,LB)
DIF LC,WRTX,APOLD
CALL PRIPUT(2HLD,LD)
DIVIDE LC,BY Y,APOLE
CALL PRIPUT(2HLE,LE)

```

```

DES LE
EVALUATELD, FOR X = 7.5
CALL PRIPUT (2HLD,LD)
EMPTY LB
DESTROY LC,LB,LD
EXP MA=3*X*Y**2*Z**3+W-A*B**2.
OUTPUTMA
COPYMA, TO MB
1 EVA MA, FOR X=2., B=10, W=16
4 OUTPUT MA, MB
2 DIFFERENTIATE MB, WITH RESPECT TO Z, AND PLACE ON MC
OUT MC
3 NAME MD, ME
OUT MD , ME
EXP MF=4*SIN(PI)*COS(X+2*PI) +3*PI
OUT MF
6 REPLACE W, IN MB, WITH MF, APO MD, EXCLUDING FUNCTION ARGUMENTS
OUT MD
REP PI, IN MD, WITH MC, AND PLACE ON ME
OUT ME
EXP L1=D**3
OUT L1
EXP L4=A+B+C
ADD L4, TO L2
OUT L2
SUBL2, FOR D, IN L1, APO L3, EFA
OUT L3
SUB L2, FOR D, IN L1, APO L4, ATO A,3., B,3
OUT L4

```

```

X    LA = X**6
X    LB = 15*A*X**2 + 20*A
S    LB = LA + LB
    OUT LB
X    LA = 12*A*X
S    LB = LB - LA
    OUT LB
    COPY LB, TO LD
    OUT LD
S    LA = LB/A
    OUT LA
S    LA = LA * 2
    OUT LA
S    LB =LA*.5
    OUT LB
S    LC = LB*LA

```

```

OUT LC
IF LD,EQUAL LC, 10
OUT LD
OUT LC
10 DESTROY LD,LA,LB
GCO X**2,IN LC, APO LA
OUT LA
GET POWER OF A,IN LA, APO PWR
PRINT 11,PWR
11 FORMAT(1H1,* PWR = *F10.5//)
X LPOW = 1.
CALL NUMPY(LPOW,PWR)
OUT LPOW
DES LA
DIFFERENTIATE LC,WRT X, APO LA
OUT LA
X LB = COS(2*B + Y)
OUTPUT LB
MULTIPLY LA, BY LB,APO LD
OUT LD
EVALUATE LD, FOR B = 1.
OUT LD
EVA LD,FOR Y=-2.
OUT LD
COPY LD, TO LE
EVA LD, FOR X= 1.,A = 2.
OUT LD
EMPTY LD
ADD LE, TO LD
OUT LD
REPLACE A , IN LC, WITH LB, APO LE
OUT LE
X LF = Y
REP B, IN LE, WITH LF, APO LG,EFA
OUT LG
DES LE, LG, LF
K = 10
NAME LEX,MEX
NSIG = -1
NFAC = 1
DO710 J=1,K,2
NSIG = NSIG*(-1)
IF (J .NE. 1) NFAC = (NFAC+2)*(NFAC+1)*NFAC
FJ = J
FAC = NFAC
FF = NSIG*(1./FAC)
CALL PUTLST(LEX,FF,1HY,FJ,LEX,LEX)
710 OUT LEX

NFAC = 1
DO 20 J=1,K
FJ = J - 1
IF (FJ .NE. 0) GO TO 19
CALL PUTLST(MEX,1.,MEX,MEX)
GO TO 20
19 CONTINUE
IF (J .NE. 1) NFAC = NFAC*J
FF = 1./NFAC

```

```
CALL PUTLST(MEX,FF,1HZ,FJ,MEX,MEX)
20 OUT MEX
SUB LEX,FOR Z, IN MEX,APO LEXSIN,EFA
OUT LEXSIN

END
```

References

- [1] Lapidus, Arnold and Goldstein, Max.
Some Experiments in algebraic manipulation by
computer. Comm. ACM 7 (Aug. 1965), 501-508.
- [2] Weizenbaum, J. "Symmetric list processor",
Comm. ACM 6 (1963), 524.
- [3] Proceedings of the ACM Symposium on Symbolic
and Algebraic Manipulation, Comm. ACM 9 (Aug. 1966).
- [4] Sammet, Jean E. "An annotated descriptor based
bibliography in the use of computers for non-
numerical mathematics". Comp. Rev. 7 (July 1966)
page B-1.

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

- A. Makes any warranty or representation, express or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or
- B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

MAR 6 1968
DATE DUE

DATE DUE

~~Main 5/25/10 SK~~

GAYLORD

PRINTED IN U S A

NYU c.1
NYO-
1480-88 Lapidus

NYU c.1
NYO-

NYU c.1
NYO-
1480-88 Lapidus

AUTHOR

A computer program for
TITLE doing tedious algebra
(SYMB00)

DATE DUE	BORROWER'S NAME	ROOM NUMBER
	<i>Main sent 5/25/70</i>	

N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012

